Adapting Safety Requirements Analysis to Intrusion Detection

Robyn R. Lutz*
Jet Propulsion Laboratory, and
Department of Computer Science
Iowa State University
rlutz@cs.iastate.edu

Several requirements analysis techniques widely used in safety-critical systems are being adapted to support the analysis of secure systems. Perhaps the most relevant system safety technique for Intrusion Detection Systems is hazard analysis. Hazard analysis identifies and analyzes hazards (states that can lead to an accident) in terms of their severity of effects and likelihood of occurrence. As Rushby points out, safety engineering techniques focus on the consequences to be avoided and explicitly consider the system context [9]. Intrusions are one such class of hazards to be avoided, and can only be understood within the context of the operational system (including both legitimate users and attackers).

Fault Tree Analysis (FTA) is often used to support the hazard analysis of safety-critical systems [5, 10]. Software Fault Tree Analysis (SFTA), a refinement of FTA, offers a way to explore intrusion scenarios in support of requirements analysis for Intrusion Detection Systems [3]. SFTA can assist in deriving requirements for the software agents that must identify and respond to intrusions.

From experience with safety-critical systems, we can identify four open issues in the application of these analysis techniques to intrusion scenarios:

- Requirements completeness.
  Fault tree techniques are limited by our ability to envision the possible hazards or intrusions and the paths to those intrusions. This limitation has been addressed in safety-critical systems by integrating the backward search in SFTA with a forward search such as Failure Modes and Effects Analysis [6, 7, 8]. The advantage of the forward search is that it systematically considers the effect of each data item and each step in the process being corrupted in specific ways (e.g., a signal arriving too soon or a process hanging at a particular point). Forward search can assist in identifying new or previously unrecognized vulnerabilities in the system.

- Formal specification of assumptions.
  Validation of the assumptions on which the requirements are based is essential in intrusion scenarios. Formal specification of the assumptions facilitates this validation. In particular, incorrect assumptions about the environment, about the constraints on the operational use of the system, or about the user's good will, skill, or access can jeopardize systems. To meet the goal of assembling and reusing a library of intrusion scenarios (perhaps a collection of fault trees) we need a way to verify that the correctness of the underlying assumptions is preserved (e.g., that a trusted host still retains that status). Formally specifying vulnerable assumptions in the requirements supports such verification [2].

- Evolution of requirements.
  Intrusion detection is complicated by the dynamic nature of the distributed systems that must be monitored. Adding to the difficulty is the large quantity of data and traffic that must be sifted, often in real-time, to detect an intrusion. Consequently, we must plan for the requirements on the intrusion detection software to be rapidly evolving and supportive of mobilility. Runtime monitoring is central to the detection of intrusion scenarios, since it can profile usage and identify evolving conditions that may threaten the security of the system. In addition, runtime monitoring can be combined with goal-based reasoning about requirements and strategies for reconciling deviations of the runtime behavior from requirements [1].

*The work described in this paper was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, under a contract with the National Aeronautics and Space Administration. Partial funding was provided under NASA's Code Q Software Program Center Initiative UPN 323-08.

- Requirements-based architectural reuse.
  The need for coordinated responses to attacks encourages the identification of architectures that are closely tied to the security requirements of a system. The relationships between architectural attributes (e.g., adaptability) and security properties are of particular interest. Given that many intrusions share common stages, steps, and processes, the opportunity for architectural-based reuse appears to merit further investigation.

The use of Software Fault Tree Analysis can assist in the identification, analysis, documentation, and reuse of intrusion scenarios. Requirements for the software agents that must detect the dangerous states and respond to the intrusions can be derived from the resulting descriptions of the system's behavior. The software is verified against these intrusion scenarios as it is designed and implemented [4]. The delivered system can then be validated against the security requirements derived from the SFTA, and the operational system maintained based on the updated requirements derived from the evolving SFTA.

References

[1] Feather, M. S. and S. Fickas and A. van Lamsweerde and C. Ponsard, "Reconciling Systems Requirements and Runtime Behavior," *Proc 9th IEEE Int Workshop on Software Specification and Design* , 1998.

[2] Hansen, K.M. and A. P. Ravn and V. Stavridou, "From Safety Analysis to Software Requirements," *IEEE Trans on Software Eng*, 24 (7), 1998, pp. 573–584.

[3] Helmer, Guy, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, and Robyn Lutz, "Software Fault Tree and Colored Petri Net Based Specification, Design and Implementation of Agent-Based Intrusion Detection Systems," submitted.

[4] Knight, John C. and Luis G. Nakano, "Software Test Techniques for System Fault-Tree Analysis," *Proc of 16th Int Conf on Computer Safety, Reliability, and Security* , 1997.

[5] Leveson, Nancy, *Safeware*, Addison-Wesley, 1995.

[6] Lutz, Robyn R. and Robert Woodhouse, "Requirements Analysis Using Forward and Backward Search," *Annals of Software Eng* , 3, 1997, pp. 459–475.

[7] Maier, Thomas, "FMEA and FTA To Support Safe Design of Embedded Software in Safety-Critical Systems," *Proc CSR 12th Annual Workshop on Safety and Reliability of Software Based Systems*, 1995.

[8] McDermid, John A. and M. Nicholson and D. J. Pumfrey and P. Fenelon, "Experience with the application of HAZOP to computer-based systems," *Proc 10th Annual Conf on Computer Assurance* , 1995, pp. 37–48.

[9] Rushby, John, "Critical System Properties: Survey and Taxonomy", *Reliability Engineering and System Safety*, 43 (2), 1994.

[10] Sullivan, Kevin and Joanne Bechta Dugan and David Coppit, "The Galileo Fault Tree Analysis Tool," *Proc 29th Annual IEEE Int Symposium on Fault-Tolerant Computing*, 1999.